REMARKS

Claims 1, 15, and 26 have been amended. Claims 1-5, 7, 8, 10-19, 21, 23-30, 32, and 34-36 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Claim Objections:

The Examiner objected to an informality caused by Applicant's inadvertently inverting the order of two words in claim 1. As requested by the Examiner, the claim has been amended to correct the error.

Section 103(a) Rejections:

The Office Action rejected claims 1, 2, 4, 5, 7, 8, 10-12, 14-16, 18, 19, 21, 23, 25-27, 29, 30, 32, 34 and 36 under 35 U.S.C. § 103(a) as being unpatentable over Claussen et al. (U.S. Patent 6,732,330) (hereinafter "Claussen") in view of Murren et al. (U.S. Patent 7,000,185) (hereinafter "Murren"), claims 3, 17 and 28 as being unpatentable over Claussen and Murren and further in view of Yu et al. (U.S. Publication 2004/0090458) (hereinafter "Yu"), and claims 13, 24 and 35 as being unpatentable over Claussen and Murren and further in view of Santos (U.S. Patent 7,107,521). Applicant respectfully traverses these rejections for at least the reasons stated below.

As a preliminary matter, Applicant notes that the Examiner ignores the requirement specified in Applicant's claim 1 for a ***software documentation generator*** to perform the recited inputting, analyzing, extracting, aggregating, and transforming. Claussen does not mention a software documentation generator. Instead, Claussen's page-handling mechanism supports the use of multiple scripting languages in a web page by marking the start and end of various scripting language code blocks in the markup for the web page [column 3, lines 13-17]. As shown in detail below, Claussen actually has very little relevance, if any, to Applicant's claimed invention. Claussen's invention is a

page-handling mechanism and runtime engine that runs on a web server [Fig. 1; column 3, lines 66-67]. It is designed to dynamically serve web page content [column 1, lines 16-18; column 4, lines 56-59]. It is decidedly *not* a ***software documentation generator for a software program***, as recited in Applicant's claim 1. Applicant notes that the words "document" and "documentation" have different meanings, and are not interchangeable, and that a ***software documentation generator for a software program*** is not the same thing as a general *document* generator. To further clarify this distinction, claim 1 has been amended to recite that the software documentation sets document the functionality of the software program.

Claussen, whether considered alone or in combination with the other cited art, does not teach or suggest generating ***software documentation*** for a software program where the software documentation documents the functionality of the software program. Claussen is directed to identifying scripting language code blocks embedded into the markup language for a web page [Abstract]. The Claussen mechanism enables use of multiple scripting languages in a Java™ server page, and it allows a developer to embed one scripting language within another [column 10, lines 19-21]. Claussen's inventive page handling mechanism 32 ***processes an HTTP page request*** and generates a response by feeding data into an output stream. The inventive functionality of Claussen's routine may be part of the integral web server program, or may stand alone [Fig. 1, element 32; column 4, lines 15-23]. In either case, **Claussen's invention processes HTTP page requests from clients** [Fig. 1, element 32; column 4, lines 15-23; column 5, lines 13-15], **compiling a servlet to serve the requested page to the client** [column 5, lines 13-15].

As a further preliminary matter concerning the general operation of Claussen's routine, Applicant observes that it employs extensible Markup Language (**XML**), which allows creation of customized markup language, with XML *tags* defining the meaning of page elements. In XML, each web page document and each of its elements is an object, with a logical structure specified in a Document Type Definition (**DTD**). A DTD is used to define a *grammar for a set of tags* for the document so that a given application may validate the proper use of the tags. In addition, **an XML document's *internal data***

structure representation **is a Document Object Model (DOM), by definition**. The DOM makes it possible to address a given XML page element as a programmable object. It is basically a tree of all the nodes in an XML file. *See* column 1, lines 20-46. Claussen's routine examines an XML document's internal data structure representation (DOM) to identify any nodes that identify a given code block. Upon encountering a node that identifies a given code block, the DOM is adjusted to account for the script code within the given code block. [Abstract; column 3, lines 18-20].

The Examiner asserts that Claussen teaches a ***software documentation generator*** *configured to input a plurality of sources related to a software program, where the plurality of sources includes different types of sources, including a software library* ***documentation*** *file for the software program*. As already explained, Claussen does *not* teach a software documentation generator. Instead, Claussen's invention is a page-handling mechanism and runtime engine that runs on a web server [column 3, lines 66-67]. Claussen's page-handling mechanism [Fig. 1, element 32] processes HTTP page requests [column 4, lines 15-17]. Page-handling mechanism 32 [Fig. 1] inputs a flat web page file and processes it to generate a servlet [Fig. 2; column 4, lines 44-45], first preprocessing the flat file into XML code [Fig. 2, step 204; Fig. 3; column 4, lines 53-54], then transforming the XML into a DOM internal data structure representation [Fig. 2, step 206; column 4, lines 54-65]. The DOM internal data structure representation of the XML document is interpreted along with its namespaces to produce the servlet [Fig. 2, step 208; Fig. 4; column 5, lines 1-5]. Claussen's three steps of preprocessing of the flat file, transforming the XML into a DOM internal data structure representation, and interpreting the DOM internal data structure representation, all serve to *generate a web page template* [column 5, lines 5-8], and they all occur at page translation time [column 5, lines 11-13]. Claussen, whether considered alone or in combination with the other cited art, emphatically does not teach a ***software documentation generator*** *inputting a plurality of different types of sources related to a software program*.

Claussen interprets the DOM internal data structure representation of an XML document along with its namespaces to produce a servlet [Fig. 2, step 208; Fig. 4; column

5, lines 1-5]. Figure 4 describes the steps involved in processing an XML document's internal data structure representation (DOM) where there are custom tags. Claussen's routine verifies the XML document's internal data structure representation (DOM), adds servlet generation variables to the root element of the DOM for later handling, and gathers all jsp:directive.page tags to ensure a consistent state. At step 408, the jsp tag libraries (which provide support for JSP 1.0 mechanisms) are registered with the root element. For example, custom tags are registered through an XML <taglib="tag-library.xml"> tag, according to the JSP 1.0 specification. This is done with custom tags to organize them and to prevent naming collisions. According to the Claussen's invention, a tag library, or "taglib," is used for the purpose. The tag library is preferably specified by a URI (Uniform Resource Identifier, a compact string of characters used to identify or name a resource on the Internet), and comprises a page identifying the tag namespace and listing the tags recognized in the namespace as well as the directives on how to load the appropriate tag handlers. In the tag-library-xml file, the name of the custom tag is listed, as well as a Java™ object name, or a URL identifying an XSL stylesheet. *See* Fig. 4 and column 5, lines 41-66.

The Examiner asserts that Figure 4 and its accompanying text at column 5 teaches teach a ***software documentation generator*** *inputting a plurality of different types of sources related to a software program, including* ***a software library documentation file for the software program***. As the foregoing discussion makes clear, the Examiner's assertion is in error. Applicant notes that the words "document" and "documentation" are different, and are not interchangeable. A ***documentation* file** for a software library for a software program communicates, i.e., *documents*, information relating to a software library. **Claussen does not teach documentation files at all**. Neither does Claussen teach documentation files for software libraries, nor inputting them into a software documentation generator. The Examiner refers to the tag library described in Claussen, apparently confusing it with a software library documentation file. A tag library is known in the art to provide a web page author with a library of preexisting custom tags [column 2, lines 23-24], which may be defined for customized markup language, with XML tags defining the meaning of web page elements [column 1, lines 20-46]. At

column 5, starting at line 40, Claussen describes tag libraries, which are used to catalog and register custom tags. **Claussen's library of markup language tags has nothing to do with software library documentation files.**

The Examiner asserts that Claussen teaches a ***software documentation generator*** *configured to input a plurality of sources related to a software program, where the plurality of sources includes different types of sources, including a software library documentation file for the software program* ***and software program source code for the software program***. As already explained, Claussen does *not* teach a software documentation generator at all. Instead, Claussen's invention is a page-handling mechanism and runtime engine that runs on a web server [column 3, lines 66-67]. In particular, Claussen does not teach *a* ***software documentation generator*** *inputting* ***software program source code for the software program***.

Further in regard to claim 1, Claussen in view of Murren neither teaches nor suggests *a* ***software documentation generator*** *analyzing its input sources to identify a type for each source*. The Examiner cites Claussen's step 308 from Fig. 3 as teaching this aspect of Applicant's claim. However, Figure 3 is entirely devoted to explicating step 204 of Figure 2, namely preprocessing a flat file into XML code [column 4, line 66]. The input stream comprising the flat file is broken into tokens, and the stream is examined at step 308 of Figure 3 to determine whether there are any more tokens to process. This has nothing whatsoever to do with *a* ***software documentation generator*** *analyzing its input sources to identify a type for each source.*

Claussen in view of Murren neither teaches nor suggests *a* ***software documentation generator*** *extracting information from its plurality of input sources based on the type of the source, where the* ***software documentation generator*** *includes multiple different input source plug-ins, each input source plug-in corresponding to a respective one of the input source types, and each configured to extract information from sources of a type to which it corresponds*. The Examiner asserts that Figs. 3-4 teach *a* ***software documentation generator*** *extracting information from its plurality of input sources based*

on the type of the source. However, as discussed earlier, Fig. 3 illustrates preprocessing a flat file into XML code, and Fig. 4 depicts interpreting the XML document's internal data structure representation (DOM), along with its namespaces, to produce a servlet [column 5, lines 1-5]. This has nothing to do with a **software documentation generator**, much less one *configured to extract information from its plurality of input sources based on the type of the source*.

The Examiner asserts that Claussen teaches *a **software documentation generator** which includes an input source plug-in* (but not a *plurality* of *different* input source plug-ins) *configured to extract information from input sources to the software documentation generator*. It is already established that Claussen does not teach a ***software documentation generator***, much less one which includes *an input source plug-in for extracting information from sources of a type to which it corresponds*. Examiner cites steps 419-422 of Figure 4. In fact, there is no step 419 and there is no step 421 in Figure 4. Steps 420 and 422 depict the invocation of a tag handler and the registration of new tag libraries, respectively. Both actions are part of the interpreting of the XML document's internal data structure representation (DOM), along with its namespaces, to produce a servlet. Nothing in Figure 4 has any bearing on *a **software documentation generator** which includes an input source plug-in configured to extract information from input sources to the software documentation generator.*

The Examiner also cites text in Claussen associated with Figure 6. The cited text beginning at column 7, line 64, and ending at column 8, line 14, describes the operation of a tag handler, a process used to hand off execution (of a custom tag in the XML document's internal data structure representation) to some other process, e.g., a Java™ object (through a custom tag interface), or via an XSL style sheet [column 6, lines 31-35]. This, too, has no bearing on *a **software documentation generator** which includes an input source plug-in configured to extract information from input sources to the software documentation generator.* Examiner additionally cites Figure 7, which illustrates translating an element of an XML document's internal data structure representation (DOM) into a text macro expansion, which is then converted into a replacement element

for the original element of the XML document's internal data structure representation (DOM). Again, this has no relevance at all to *a **software documentation** **generator** which includes an input source plug-in configured to extract information from input sources to the software documentation generator.* Examiner again cites step 422 of Figure 4, which concerns the registration of new tag libraries in the tag library registry, as teaching *a **software documentation** **generator** which includes an input source plug-in configured to extract information from input sources to the software documentation generator, where the input sources are of a type to which the plug-in corresponds.* As already explained, Figure 4 is wholly directed to interpreting an XML document's internal data structure representation (DOM), along with its namespaces, to produce a servlet to serve a requested Web page in response to an HTTP client request. The registration of new tag libraries depicted at step 422 of Figure 4 has nothing to do with *a **software documentation** **generator's** input source plug-ins configured to extract information from input sources to the software documentation generator, where the input sources are of a type to which the plug-in corresponds.*

Further in regard to claim 1, Claussen in view of Murren neither teaches nor suggests *a **software documentation** **generator** configured to aggregate information extracted from input sources to the software documentation generator into a uniform format, where the input sources are related to a software program.* Examiner cites Claussen's processing of an XML document's internal data structure representation (DOM) with custom tags (column 5), and Claussen's translation of a flat file into an XML internal data structure representation (DOM) (columns 16-17), as teaching this aspect of Applicant's claim. However, as has already been shown, Claussen does not teach or suggest a ***software documentation** **generator for a software program***, much less one *configured to aggregate information extracted from input sources to the **software documentation** **generator** into a uniform format, where the input sources are related to a software program.*

Further in regard to claim 1, Claussen in view of Murren neither teaches nor suggests *a **software documentation** **generator for a software program** configured to*

transform aggregated information extracted from input sources to the ***software documentation generator*** *into one or more specified sets of software documentation for the software program, where the* ***software documentation generator*** *includes a plurality of different transformer plug-in sets, such that each transformer plug-in set corresponds to one or more respective types of* ***output software documentation sets****, and each transformer plug-in set is configured to generate one or more respective* ***output software documentation sets*** *of types to which the transformer plug-in corresponds, such that the specified sets of software documentation document the functionality of the software program.* Examiner refers to Figure 5 (illustrating tag handling that occurs during the interpreting of an XML document's internal data structure representation (DOM), along with its namespaces, to produce a servlet), and Examiner also refers to the multiplicity of scripting languages found within a single web page in Claussen, asserting that these aspects of Claussen teach *a* ***software documentation generator for a software program*** *configured to transform aggregated information extracted from input sources to the* ***software documentation generator*** *into one or more specified sets of software documentation for the software program.* As previously shown, Claussen's tag handling has no bearing on *a* ***software documentation generator for a software program*** as recited in claim 1. The presence within Claussen's single web page of code blocks written in multiple scripting languages also has no bearing on *a* ***software documentation generator for a software program generating*** *one or more specified sets of software documentation for the software program.* In fact, Claussen does not teach a ***software documentation generator for a software program***, nor one *configured to transform aggregated information extracted from input sources to the* ***software documentation generator*** *into one or more specified sets of software documentation for the software program, such that the specified sets of software documentation document the functionality of the software program.*

Examiner refers to Claussen's "new document" returned by both the tag handler at step 510 of Figure 5, and the tag handler for Figure 6, step 618. However, both step 510 of Figure 5 and step 618 of Figure 6 refer to the page template generated via DOM translation of the original flat file [step 208 of Figure 2 depicts interpreting the XML

document's internal data structure representation (DOM), along with its namespaces, to produce the servlet by generating a page template (column 5, lines 1-8); step 208 of Figure 2 is illustrated in more detail in Figure 4, and step 420 of Figure 4, depicting the invocation of a tag handler, is illustrated in yet more detail in Figures 5 and 6 (column 6, lines 51-52); *see* column 7, lines 59-62, and column 8, lines 40-41]. Examiner also refers to column 25, lines 49-61, which describes Claussen's use of custom tags to replace script code in authored web pages, keeping the page clean and easy to maintain, and allowing the script code to be kept separately, so that it only needs to be debugged once. Contrary to the Examiner's assertion, these portions of Claussen do not teach *a software documentation generator for a software program configured to transform aggregated information extracted from input sources to the software documentation generator into one or more specified sets of software documentation for the software program, where the* ***software documentation generator includes a single transformer plug-in set*** *(rather that a plurality of different transformer plug-in sets) such that the transformer plug-in set* ***corresponds to one or more respective types of output software documentation sets***. Examiner again apparently refers to column 25, lines 49-61, as teaching that *each transformer plug-in set of the* ***software documentation generator*** *is configured to* ***generate*** *one or more respective* ***output software documentation sets*** *of types to which the transformer plug-in corresponds*. Again, the cited lines describe Claussen's use of custom tags to replace script code in authored web pages, to keep the page clean and easy to maintain, and to allow the script code to be kept separately, so that it only needs to be debugged once. The cited portion of Claussen has no bearing on Applicant's claim 1.

Examiner admits that Claussen does not explicitly disclose *wherein the software documentation generator includes a plurality of different input source plug-ins, wherein each input source plug-in corresponds to a respective one of the source types*. Examiner also admits that Claussen does not disclose *a plurality of different transformer plug-in sets, wherein each transformer plug-in set corresponds to one or more respective types of output software documentation sets and each transformer plug-in set is configured to generate one or more respective output software documentation sets of types to which the*

plug-in corresponds. To remedy these admitted deficiencies in Claussen, Examiner appeals to Murren.

Applicant again notes that the Examiner ignores the requirement specified in Applicant's claim 1 for a ***software documentation generator*** to perform the recited inputting, analyzing, extracting, aggregating, and transforming. Murren, like Claussen, does not mention a software documentation generator. Murren's invention customizes server-executable web pages [column 1, lines 6-8]. Murren's customization system "compiles" a computer program (e.g., Java Server Page [JSP] or Active Server Page [ASP]) by identifying the content of each statement (e.g., tag) of the computer program that may be customized. The customization system identifies the type (e.g., Java Server Page [JSP] tag) associated with each statement. Based on the type of statement, the customization system identifies content of the statement that can be customized and stores the identified content in a custom content bundle. A custom content bundle (e.g., resource bundles provided for in the Java platform) contains name and value pairs that map content identifiers to the corresponding content. The customization system replaces the identified content in the statement with an include content command (e.g., an include content tag when the program is a Java Server Page [JSP]) that includes the content identifier of the corresponding content. When the computer program is executed, the include content command of the statement causes the retrieving of a content associated with its content identifier from a custom content bundle. *See* column 2, column 55 to column 3, line 15. Murren's system clearly is not a ***software documentation generator for a software program***, as recited in Applicant's claim 1. Murren does not teach or suggest a software documentation generator for a software program. Murren teaches customizing server-executable web pages. Applicant notes again that the words "document" and "documentation" have different meanings, and are not interchangeable, and that a ***software documentation generator for a software program*** is not the same thing as a general *document* generator. To further clarify this distinction, claim 1 has been amended to recite that the software documentation sets document the functionality of the software program.

The Examiner refers to Murren's tag processing and execution of code for server page (JSP) tags to generate portions of a customized HTML page (Fig. 7). This portion of Murren has nothing to do with *a **software documentation** generator extracting information from a plurality of different input sources based on the type of the source, where the **software documentation** generator includes a plurality of different input source plug-ins, each input source plug-in corresponding to a respective one of the input source types.* Neither does it have anything to do with *a **software documentation generator for a software program** configured to transform aggregated information extracted from input sources to the **software documentation** generator into one or more specified sets of software documentation for the software program, where the **software documentation** generator includes a plurality of different transformer plug-in sets, such that each transformer plug-in set corresponds to one or more respective types of **output software documentation sets**, and each transformer plug-in set is configured to generate one or more respective **output software documentation sets** of types to which the transformer plug-in corresponds.*

The Examiner asserts that it would have been obvious "to implement the plugins by Murren (plug-ins - col. 4 lines 7-19, lines 28-40) so that plug-ins for handling a source and to generate output document exist in plurality thereof where each is instantiated for the corresponding type as evidenced from above because this would obviate writing APIs from scratch or downloading or compiling code, in alleviating undue exertion of effort and resources during browser application runtime as purported above (see Claussen: plug-ins col. 4 lines 28- 40) whereby plug-ins usage would be a known practice for its being expedient usability." The first cited portion of Murren at column 4 describes Fig. 1A, the **preprocessing** of JSP source by one translator to generate an XML version of the source JSP, and by another translator converting the resultant XML version of the source JSP into a DOM version. The second cited portion of Murren at column 4 describes Fig. 1B, and the compilation and execution of a Java Server Page (JSP), including HTML and custom tag compiling. The Examiner apparently asserts that these portions of Murren at column 4 teach *a software documentation generator including a plurality of different input source plug-ins, wherein each input source plug-in corresponds to a respective one*

of the source types. However, Murren does not teach a *software documentation generator* at all, much less one *including a plurality of different input source plug-ins, wherein each input source plug-in corresponds to a respective one of the source types.* Examiner's assertion is clearly erroneous. Examiner also cites Claussen's paragraph at column 4, lines 28-40, which refers to a typical Web client's suite of Internet tools, including a Web browser with a Java™ Virtual Machine (JVM) and support for application plug-ins, or helper applications. Examiner apparently asserts that Claussen's mention of plug-ins for a client's Web browser teaches plug-ins configured to extract information from input sources for the software documentation generator as recited in Applicant's claim 1. This assertion, too, is clearly erroneous.

Apparently Examiner proposes combining Murren with Claussen, but does not reveal how to combine the two references. One might infer that the Examiner proposes augmenting Claussen with the preprocessing, compiling, and executing of JSP source recited by Murren in column 4. However, Claussen unmodified already possesses the capability to do the preprocessing, compiling, and executing of JSP source. Moreover, even with these capabilities, Claussen fails to teach or suggest the full combination of limitations recited in Applicant's claim 1. Examiner refers to writing APIs from scratch, and to downloading and compiling code, and asserts that the inclusion of Murren's teachings in Claussen would eliminate the need for such writing, downloading, and compiling. The Examiner's assertion reflects Examiner's own opinion, and is not accompanied by any supporting evidence of record or any other valid reason. Examiner also alludes to alleviating undue exertion and effort and resources during browser application runtime. Examiner offers no explanation as to how the incorporation of Murren's teaching into Claussen would alleviate undue exertion and effort and resources during browser application runtime. Even if this were possible, Claussen in view of Murren would not teach or suggest the full combination of limitations recited in Applicant's claim 1. Moreover, alleviating undue exertion and effort and resources during the operation of a Web browser application has no bearing on Applicant's claim 1. Thus, one of ordinary skill would not attempt to combine the teachings of Murren with

the teachings of Claussen as proposed by the Examiner. Accordingly, the Examiner has failed to establish a *prima facie* case of obviousness.

Claims 15 and 26 both recite the generation of sets of **software documentation for the software program**, where the specified sets of software documentation **document the functionality of the software program**. In light of this, and of the other limitations recited in claims 15 and 26, it is clear from the foregoing argument that Claussen in view of Murren does not teach or suggest either claim 15 or claim 26.

Applicant also asserts that the rejection of numerous ones of the dependent claims is further unsupported by the cited art. However, since the rejections have been shown to be unsupported for the independent claims, a further discussion of the dependent claims is not necessary at this time.

CONCLUSION

Applicant submits the application is in condition for allowance, and notice to that effect is respectfully requested.

If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5681-75900/RCK.

Respectfully submitted,

/Robert C. Kowert/
Robert C. Kowert, Reg. #39,255
Attorney for Applicant

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: December 31, 2008